软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(8):1982−1998 [doi: 10.13328/j.cnki.jos.005201]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

虚拟化环境下面向多目标优化的自适应 SSD 缓存系统*

唐 震[1,2,3], 吴 恒[1,2], 王 伟[1,2], 魏 峻[1,2], 黄 涛[1,2]

[1](中国科学院 软件研究所 软件工程技术研发中心,北京 100190)

[2](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[3](中国科学院大学,北京 100049)

通讯作者: 吴恒, E-mail: wuheng09@otcaix.iscas.ac.cn

摘 要: 以 SSD(solid state drive)为代表的新型存储介质在虚拟化环境下得到了广泛的应用,通常作为虚拟机读写缓存,起到优化磁盘 I/O 性能的作用.已有研究往往关注 SSD 缓存的容量规划,依据缓存读写命中率评价 SSD 缓存分配效果,未能充分考虑 SSD 的服务能力上限,难以适用于典型的分布式应用场景,存在虚拟机抢占 SSD 缓存资源,导致虚拟机中应用性能违约的可能.实现了虚拟化环境下面向多目标优化的自适应 SSD 缓存系统,考虑了 SSD 的服务能力上限.基于自适应闭环实现对虚拟机和应用状态的动态感知.动态检测局部 SSD 缓存抢占状态,基于聚类方法生成虚拟机的优化放置方案,依据全局 SSD 缓存供给能力确定虚拟机迁移顺序和时机.实验结果表明,该方法在应对典型分布式应用场景时可以有效缓解 SSD 缓存资源的争用,同时满足应用对虚拟机放置的需求,提升应用的性能并兼顾应用的可靠性.在 Hadoop 应用场景下,平均降低了 25%的任务执行时间,对 I/O 密集型应用平均提升 39%的吞吐率.在 ZooKeeper 应用场景下,以不到 5%的性能损失为代价,应对了虚拟化主机的单点失效带来的虚拟机宕机问题.

关键词: 固态盘;缓存;虚拟化;动态迁移

中图法分类号: TP316

中文引用格式: 唐震,吴恒,王伟,魏峻,黄涛.虚拟化环境下面向多目标优化的自适应 SSD 缓存系统.软件学报,2017,28(8):1982−1998. http://www.jos.org.cn/1000-9825/5201.htm

英文引用格式: Tang Z, Wu H, Wang W, Wei J, Huang T. Self-Adaptive SSD caching system for multiobjective optimization in virtualization environment. Ruan Jian Xue Bao/Journal of Software, 2017,28(8):1982−1998 (in Chinese). http://www.jos.org.cn/1000-9825/5201.htm

Self-Adaptive SSD Caching System for Multiobjective Optimization in Virtualization Environment

TANG Zhen[1,2,3], WU Heng[1,2], WANG Wei[1,2], WEI Jun[1,2], HUANG Tao[1,2]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: As a new type of storage media, solid state drive (SSD) is widely used in virtualization environment. SSD is usually used as the read and write cache of the virtual machine (VM) storage to improve the disk I/O performance of the VMs. Existing SSD caching schemes mostly focus on the capacity planning of the SSD cache and use metrics such as cache hit rate to evaluate the effect of SSD cache

* 基金项目: 国家重点研发计划(2016YFB1000103); 国家自然科学基金(61572480); 中国科学院青年创新促进会(2015088)

Foundation item: National Key Research and Development Program of China (2016YFB1000103); National Natural Science Foundation of China (61572480); Youth Innovation Promotion Association, the Chinese Academy of Sciences (2015088)

收稿时间: 2016-06-20; 修改时间: 2016-09-21, 2016-11-11; 采用时间: 2016-12-01; jos 在线出版时间: 2017-01-12

CNKI 网络优先出版: 2017-01-12 10:36:02, http://www.cnki.net/kcms/detail/11.2560.TP.20170112.1036.006.html

allocation. Since they do not consider the limitations of service capabilities of SSD, which may lead to the contention of cache resource and the performance degradation and violations among VMs, they are not suitable to be used with some typical distributed applications. This article proposes a self-adaptive SSD caching system for multiobjective optimization in virtualization environment, to reduce the resource contention, and take the limitations of service capabilities of SSD into consideration. With the help of closed loop adaption, it can dynamically detect the status of VMs and applications. Moreover, it continuously detects the contention of SSD cache, generates the migration plan using clustering algorithm and decides the timing and order of the VM migrations according to the capabilities of SSD as well as the characteristics and requirements of applications. The evaluation shows that when facing the scenarios of using some typical distributed applications, the contention of SSD cache resource is reduced and the requirements of applications are considered, which lead to the improvement of performance and reliability of applications. For Hadoop applications, the execution time of jobs is reduced by 25% on average, and the throughput for I/O sensitive applications is improved by 39%. For ZooKeeper applications, the service outage caused by the single point of fault of the hypervisor can be handled at the cost of less than 5% of performance degradation.

Key words: solid state drive (SSD); caching; virtualization; live migration

虚拟化技术通过抽象成虚拟机的模式分时复用物理资源,既保障了应用的隔离性,又提高了资源利用率,已成为应用运行支撑的主流基础设施[1].近年来,越来越多的大数据应用实现了虚拟化的迁移,对虚拟机 I/O 的性能也提出了更高的要求[2,3].目前,以固态盘(solid state drive,简称 SSD)为代表的新型存储介质也引起了广泛关注[4,5].与传统的机械硬盘(hard disk drive,简称 HDD)相比,固态盘具有较高的顺序读写和随机读写速率,但擦写次数有限,且寿命相对较短.基于性价比考虑,在虚拟化环境下,SSD 通常用作虚拟机的磁盘缓存,以提高虚拟机的 I/O 性能[6−8].虚拟机的 I/O 请求会先通过缓存,若缓存命中,则不会再向分布式存储请求数据,从而也可以有效降低后端分布式存储的负载.

缓存命中率和工作集(在某一时间范围内访问的数据集合)大小作为缓存的关键指标,被广泛用于指导 SSD 缓存管理.例如,刻画缓存命中率及相关导出指标并将 SSD 缓存分区供多台虚拟机使用,以提升 I/O 性能[6,9];或通过改进的工作集和对应的缓存策略提高缓存资源利用率[8].然而,已有研究仍然存在不足之处,面临如下的挑战.

(1) 已有研究往往关注单个 Hypervisor(虚拟化主机),依据局部读写缓存命中率进行 SSD 缓存的容量规划,并评价 SSD 分配效果,并未全局考虑多个 Hypervisor 的 SSD 缓存供给能力,存在虚拟机抢占 SSD 缓存资源,导致虚拟机中应用性能违约的可能.由于局部 SSD 资源供给上限约束,对 SSD 缓存资源争用的状况无法通过简单改变缓存分配方案来消减,而需要通过虚拟机动态迁移的方式实现虚拟机的重新放置来解决,这也对 SSD 缓存系统提出了新的挑战.如图 1 所示是一个未考虑 SSD 缓存供给能力而导致虚拟机争用 IOPS(I/O operations per second(每秒 IO 操作数))资源的场景.3 台图片服务器部署在同一台 Hypervisor 上,它们都需要频繁读写小文件(图片),因此对 IOPS 资源有很高的需求.但同时,在一定时间段内访问到的图片总量是有限的,因此它们的工作集相对较小,对于缓存容量的需求也较小.在这一场景下,从缓存命中率或工作集大小的角度去衡量,会发现 SSD 未达到资源供给的瓶颈,但从 IOPS 角度衡量时会发现,SSD 的 IOPS 资源已被 3 台虚拟机争用,可能产生性能违约.

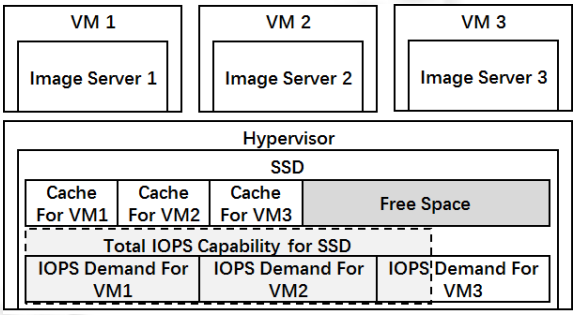

Fig.1 Example of IOPS resource contention of SSD cache

图 1 SSD 缓存的 IOPS 资源争用的例子

(2) 在进行虚拟机迁移时,同样需要考虑用户对虚拟机的使用模式,从而避免产生性能和服务质量违约.虚拟化环境下,最终用户使用虚拟机的方式通常是向虚拟化服务提供商租用一批虚拟机,并在其上部署应用从而提供服务.这些应用通常由多台虚拟机组成,这些虚拟机各司其职,相互协同,共同对外提供服务.这些虚拟机也由于从属于同一个应用的关系而天然具有某种关联,现有工作在进行缓存资源管理时并未考虑到虚拟机之间的这类关联,因而可能忽略一些至关重要的信息.在执行虚拟机动态迁移时,我们也需要考虑应用本身的特点和对虚拟机放置的倾向性,以此来指导虚拟机的迁移.

(3) 虚拟机的动态迁移操作本身开销较大,需要进行合理规划,以尽可能地降低对被迁移虚拟机以及整个虚拟机集群的性能影响.虚拟机的动态迁移操作会对迁入和迁出的物理机资源供给产生一定的影响,并占用一定的网络带宽以传输虚拟机的实时状态,同时也会在短时间内降低被迁移虚拟机的性能.错误的迁移顺序或迁移频率会导致整体性能下降,并产生不必要的开销.与之相对地,迁移频率过低也会使虚拟机的优化放置达不到应有的效果.因此,需要合理权衡虚拟机动态迁移的顺序和时机.

为了应对上述 3 个挑战,本文提出了虚拟化环境下面向多目标优化的自适应 SSD 缓存系统,在现有工作关注的容量规划问题之外,考虑了应用的性能、可靠性以及 SSD 缓存资源的负载均衡.考虑到应用负载的多变性,其在应对不同负载时的 I/O 表现不同,对 SSD 缓存资源的需求也不同,因此需要自适应机制进行持续监测和调整.此外,我们提出了一种感知应用对虚拟机放置倾向性的机制,将应用的需求化归为内聚(倾向于放置在同一 Hypervisor 上)和隔离(倾向于放置在不同 Hypervisor 上)两个维度的放置倾向性,从而指导虚拟机动态迁移.在生成具体迁移方案时,我们提出了一种基于聚类的方法,同时考虑 SSD 缓存服务能力以及应用的特点和对虚拟机放置的倾向性,生成优化的虚拟机放置方案.此外,考虑到虚拟机动态迁移本身的开销较大,我们提出了一种从虚拟机放置方案生成优化迁移顺序的方法,以此降低虚拟机动态迁移的开销.实验结果表明,针对大数据处理系统(Hadoop[10,11])和分布式应用协同系统(ZooKeeper[12,13])这两类虚拟化环境下常见的应用场景,本文的自适应 SSD 缓存系统可以有效地实现 SSD 资源的负载均衡.同时,在迁移时考虑到应用对虚拟机放置的倾向性,从而提升应用的性能和可靠性,提升 SSD 缓存资源的利用率,并降低了后端分布式存储的负载,实现了多目标优化.

本文的主要贡献包括以下 3 点:(1) 提出一种自适应的 SSD 缓存机制,可以动态地监测应用对 SSD 资源的使用情况,并对虚拟机放置进行必要的调整,实现 SSD 缓存资源的负载均衡,从而提高 SSD 缓存资源的利用率;(2) 感知应用的特点并刻画其对虚拟机放置的倾向性,进一步指导虚拟机动态迁移,以支持对应用性能和可靠性的优化;(3) 提出了构建虚拟机优化迁移顺序的方法,以降低虚拟机动态迁移的开销.我们实现了虚拟化环境下面向多目标优化的自适应 SSD 缓存原型系统,并验证了该系统在大数据处理和分布式协同场景下的有效性.

本文第 1 节介绍问题的研究背景.第 2 节分析问题的具体场景,并说明 SSD 负载均衡以及考虑应用需求的必要性.第 3 节介绍以 SSD 缓存资源负载均衡为导向的虚拟机动态迁移的具体方法,并介绍了原型系统的架构及主要模块.第 4 节给出原型系统在大数据处理和分布式协同场景下的具体表现以及结果分析.第 5 节介绍相关工作.第 6 节总结全文,并给出对未来工作的展望.

1 研究背景

虚拟化平台通常由多台部署了 Hypervisor 的物理机组成,由 Hypervisor 完成虚拟机的生命周期管理.虚拟机通常需要占用较大的磁盘空间,同时需要满足可靠性和可用性需求,因此,组成虚拟化平台的物理机通常会连接到一个共享的后端分布式存储系统,用于保存虚拟机镜像.在使用虚拟机时,会通过网络直接访问共享存储上的镜像文件.虚拟机需要直接从共享存储上运行操作系统,I/O 操作相对较为频繁,这也给分布式存储系统带来较大的压力.SSD 作为一种新型存储介质,与传统的机械硬盘以及基于网络的存储相比,其速度较快,顺序读写速度约为机械硬盘的 3 倍~5 倍,随机读写 IOPS 约为机械硬盘的 100 倍~1 000 倍[14,15],尤其适用于应对大规模的随机读写 I/O 负载,在虚拟化环境下得到了广泛的应用.SSD 通常部署在 Hypervisor 上,用作虚拟机的读写缓存,其主要使用模式如图 2 所示.合理地使用 SSD 缓存资源可以有效地提高虚拟机的 I/O 性能,并降低后端分布式存储系统的负载.

应用是虚拟化环境下的主要服务模式.应用通常由多台虚拟机组成,互相协同,共同对外提供服务.虚拟机之间会由于从属于同一个应用而天然具有一定的关联和依赖.例如,典型的事务型应用通常由多台前端 Web 服务器、中间件服务器以及后端数据库服务器组成,前端 Web 服务器会依赖于中间件服务器提供业务能力支撑,也依赖于数据库服务器提供持久化支持;在大数据处理(如 Hadoop)和分布式协同(如 ZooKeeper)应用中,虚拟机具有对称性,且共同应对相似的工作负载,即用户自定义的大数据处理逻辑.这些具有关联特性的虚拟机对 SSD 缓存的需求具有一定的特征,可以通过感知应用及其工作负载的方式得到.应用内的虚拟机之间具备的关联值得特别关注,它对于指导 SSD 缓存的资源管理具有重要的意义.

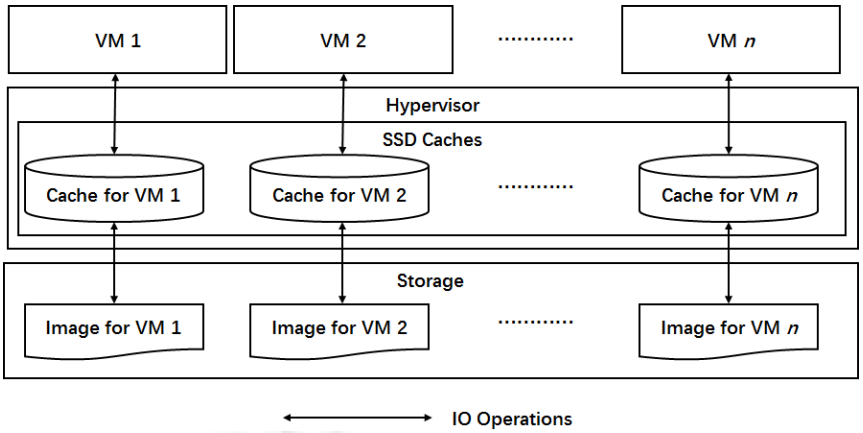

Fig.2 Usage of SSD caching in virtualization environment

图 2 虚拟化环境下 SSD 缓存的应用模式

虚拟机的动态迁移是 Hypervisor 的重要特性之一,这一特性允许虚拟化服务提供商在运行时动态调整虚拟机的放置,从而实现物理机上特定资源的负载均衡,提高资源利用率.此外,虚拟机的动态迁移也可以实现集群的故障转移和热备份,从而提高虚拟机的可靠性和可用性.

值得注意的是,对用户而言,虚拟化服务提供商仅提供了简单申请虚拟机的方式来支持用户的应用部署需求.租用虚拟机的操作本身是透明的.出于对集群管理的需求,用户无法决定虚拟机的放置位置,也无法要求虚拟化服务提供商按照用户自定义需求进行虚拟机的租用操作.因此在某些场景下,不合理的虚拟机动态迁移操作会影响服务提供商所承诺的服务质量,可能导致服务违约,从而影响用户体验.在进行虚拟机动态迁移时需要合理地选择指标,同时需要考虑到用户及其在虚拟机上部署的应用的需求,从而有效消减虚拟机动态迁移带来的服务质量下降.

2 问题分析

SSD 用作虚拟机的读写缓存,可以有效地提升虚拟机的 I/O 性能.然而,当前针对 SSD 缓存资源管理的相关工作主要关注 SSD 缓存的容量规划,通常从缓存的单一指标,如缓存命中率或工作集大小等出发,以此驱动 SSD 缓存资源的分配和调整.如,以缓存命中率为主要衡量指标的方法通过调整缓存大小并监测命中率或相关指标,最终达到同一 Hypervisor 上的缓存命中率稳定且较高.由于缓存命中率直接关系到虚拟机的 I/O 性能,这些方法最终可以达到提升 I/O 性能或提高缓存利用率的目的.值得注意的是,这些方法关注了缓存本身的特性,但忽略了 SSD 本身作为存储设备的特点,即 SSD 的带宽和 IOPS 的服务能力是存在上限的.以往以缓存相关指标为中心的工作通常假设 SSD 的平均延迟是固定的,因此可由以下公式计算出虚拟机内观察到的 I/O 响应时间 L:

$$L=l_{SSD}\cdot hr+l_{Storage}\cdot(1-hr),$$

其中,l_{SSD} 代表 SSD 的平均 I/O 操作延迟,hr 代表 SSD 缓存命中率,$l_{Storage}$ 代表后端存储的平均 I/O 操作延迟.直

观上说,调整缓存大小以使缓存命中率趋于稳定,并使得由 SSD 缓存服务的所有虚拟机的平均缓存命中率最大,即可将虚拟机的 I/O 响应时间调整到最优.然而,这一方法假设缓存的服务能力是稳定且无限的,制约缓存的唯一要素是缓存大小以及与此相关的缓存命中率.这一假设在 CPU 片上缓存或内存缓存的场景下是合理的,但在 SSD 缓存的场景下存在局限性.对一些工作集较小的 I/O 密集型应用而言,将它们部署在同一台 Hypervisor 上并不会导致需求的缓存容量超过 SSD 总容量,其缓存命中率也是相对稳定且较高的,但由于应用的 I/O 特性,导致同时存在大规模的 I/O 操作.当这些 I/O 操作超过 SSD 所能提供的最大的带宽和 IOPS 时,就会对 SSD 缓存产生资源争用,导致 I/O 操作在 SSD 一层排队,从而对平均响应时间造成较大的影响.应用之间也会由此产生 I/O 性能干扰.这一干扰是由资源争用引发的,并不能通过调整 SSD 缓存分配来消减,需要通过全局角度考虑 SSD 缓存资源供给能力并进行负载均衡来解决.这也对 SSD 缓存资源管理提出了新的挑战,亟需一种虚拟化环境下面向多目标优化的 SSD 缓存系统,在容量规划之外权衡 SSD 缓存的资源供给能力,在检测到虚拟机对 SSD 缓存需求超过其供给能力时将部分虚拟机迁移到其他 Hypervisor 上,以此缓解资源争用并实现 SSD 缓存的负载均衡,提高 SSD 缓存资源的利用率.

如前文所述,应用是虚拟化环境下的主要服务模式,虚拟机之间由于从属于同一应用会产生一些天然的关联.在执行由 SSD 缓存负载均衡导向的虚拟机动态迁移时,需要考虑应用的特点和对虚拟机放置的倾向性以及虚拟机之间的关联,从而兼顾应用对性能和可靠性的需求.

对应用而言,在虚拟机放置方面可以化归为两类需求.

(1) 一类应用出于性能考虑,倾向于将虚拟机内聚,即,部署在同一台 Hypervisor 上.如运行在 Hadoop 上的部分大数据处理应用,存在对 HDFS(Hadoop distributed file system(Hadoop 分布式文件系统))上的文件的强依赖和数据本地性需求,即运行过程中需要频繁读写存储于 HDFS 上的数据.在此场景下,若能将这些虚拟机部署在同一台 Hypervisor 上,则可以消减网络传输瓶颈,并使虚拟机受益于 Hypervisor 内部的虚拟机优化机制.

(2) 另一类应用出于可靠性或负载均衡的需求,倾向于将虚拟机隔离,即部署在不同的 Hypervisor 上.如以 ZooKeeper 为代表的分布式协同服务,其关键目标是提供可靠的协同机制,为用户维护运行时状态等信息.这类应用虽然也提供存储服务,但它们对可靠性的需求远大于对性能的需求.这类应用中的虚拟机倾向于部署在不同的 Hypervisor 上,从而使得单一 Hypervisor 失效时剩余的节点仍然能够正常提供服务.对 HDFS 而言,倾向于将元数据服务器和备份元数据服务器放置在不同的 Hypervisor 上,以应对单点失效问题.

值得注意的是,应用的行为是动态变化的,它们所承载的工作负载也是动态变化的,这也导致应用的行为以及它们对虚拟机内聚和隔离的需求无法提前预测,需要通过运行时的动态监控分析来获得.因此,静态的虚拟机放置和 SSD 缓存资源分配方法无法满足需求.这也对虚拟化环境下面向多目标优化的 SSD 缓存系统提出了挑战,亟需一种具备自适应能力的 SSD 缓存系统,持续监测应用的状态,感知应用对虚拟机放置的倾向性,并在执行由 SSD 缓存负载均衡导向的虚拟机动态迁移时考虑这一倾向性,以尽可能地提升应用的性能和可靠性.

此外,虚拟机的动态迁移本身是一个开销较大的操作.动态迁移的基本原理是:将指定虚拟机的内存数据持续复制到目标 Hypervisor 上,并在完成复制之后短暂挂起虚拟机,在目标 Hypervisor 继续运行.对 SSD 缓存系统而言,还需要对虚拟机绑定的 SSD 缓存进行动态迁移,处理脏数据并在目标 Hypervisor 上重新构建缓存.因此,动态迁移操作会对被迁移的虚拟机的性能产生一定的影响,也会在一定时间内占用较大的网络带宽,可能会对同样使用网络带宽资源的后端分布式存储系统产生影响,从而降低整个虚拟机集群的性能.这也对以虚拟机动态迁移作为主要手段的 SSD 缓存系统提出了新的挑战,在确定了新的虚拟机放置方案之后,SSD 缓存系统需要进一步合理规划虚拟机动态迁移的顺序和时机,尽可能地降低对性能的影响.

3 虚拟化环境下面向多目标优化的自适应 SSD 缓存系统

我们提出了虚拟化环境下面向多目标优化的自适应 SSD 缓存系统,以应对第 2 节中所述的 3 个挑战.这一

系统主要由一个自适应闭环驱动虚拟机监测以及虚拟机放置方案生成过程,从而可以应对动态变化的应用行为和工作负载.在生成虚拟机放置方案时我们使用了基于聚类的方法,主要由 SSD 缓存负载均衡驱动,同时考虑到了应用对虚拟机放置的倾向性.在生成虚拟机动态迁移方案时,我们还给出了迁移优先级规则.

3.1 自适应闭环

虚拟化环境下面向多目标优化的自适应 SSD 缓存系统由一个自适应闭环驱动.这一自适应闭环主要由监测(monitoring)、训练(training)和执行(applying)这 3 个主要模块组成.其主要原理如图 3 所示.

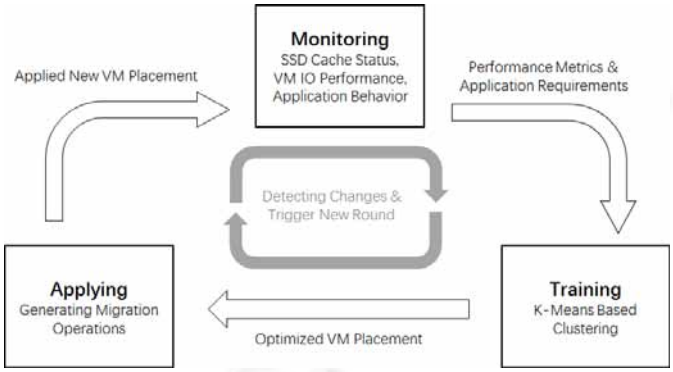

Fig.3 Overview of adaptive closed loop

图 3 自适应闭环概览

- 监测模块主要用于监测虚拟机对 SSD 缓存的使用状况以及虚拟机上部署的应用的状况,为训练模块提供数据支持.监测模块在运行时会持续监测虚拟机集群,通过部署在虚拟机和 Hypervisor 上的代理(agent)程序获得性能数据和运行时数据,并进行必要的处理.
- 训练模块主要用于加工监测模块传递的数据,并生成优化的虚拟机放置方案.我们使用基于 *K*-Means 聚类的方法实现虚拟机放置方案的生成.我们将整个虚拟机集群聚成 X 类,X 为 Hypervisor 的数量.在聚类时,我们加工了监测模块传递的有关 SSD 缓存和应用的数据,考虑到了 SSD 缓存负载均衡的需求以及 SSD 缓存供给能力的约束,并兼顾了应用对虚拟机放置的倾向性,最终给出了虚拟机的聚类方案,即虚拟机放置到不同 Hypervisor 上的方案.
- 执行模块主要用于处理训练模块生成的虚拟机放置方案,计算聚类结果与 Hypervisor 的优化映射,并进一步计算出所需的迁移操作.在此基础上,执行模块会基于降低性能开销和干扰的原则规划虚拟机迁移顺序,并最终执行虚拟机具体的动态迁移操作.

此外,监测模块会在运行时对整个虚拟机集群进行持续监测,在检测到虚拟机集群发生变化,应用边界发生变化或应用负载和放置倾向性发生突变时触发新一轮的闭环执行,从而应对动态变化的应用工作负载,实现自适应.为了控制自适应闭环调整的频率,我们设置了触发闭环执行的上下限,从而平衡虚拟机性能和虚拟机动态迁移的开销.

3.2 SSD缓存及应用状态监测

系统在虚拟机、SSD 和应用层次实现了细粒度的监测,具体原理如图 4 所示.

应用是虚拟化环境下的主要服务模式.应用通常由多台虚拟机构成,虚拟机之间会由于从属于同一个应用而具有天然关联.因此,监测模块首先需要确定整个虚拟机集群上运行的应用及应用各自所包含的虚拟机.监测模块根据虚拟机上运行的进程确定虚拟机上所运行的应用类型(如 Hadoop 或 ZooKeeper 应用等)以及虚拟机在应用中的角色(如 Hadoop 中的 HDFS 元数据节点、数据节点等,ZooKeeper 中的 Leader 节点和 Follower 节点等),之后,根据虚拟机的网络使用情况以及 TCP 连接情况确定虚拟机交互的对象.凭借这两部分可以区分出虚拟机

集群中的不同应用以及应用各自的边界,也由此确定应用内虚拟机的交互关系,从而支持对应用工作负载和虚拟机放置倾向性的进一步分析.在本文范围内,我们不考虑虚拟机集群中两个应用间出现引用的情况.

在完成应用边界及应用内虚拟机的交互之后,监测模块会从 SSD 缓存资源使用状况和应用需求两方面进行细粒度的监测.在 SSD 缓存资源使用状况方面,监测模块会收集虚拟机的 I/O 使用率,包括 I/O 操作所占时间、读写带宽以及读写 IOPS,以及 SSD 缓存本身的特性,包括分配给虚拟机的缓存大小、缓存使用率(缓存中已使用的数据块的比例)、读写命中率.这些信息有助于确定虚拟机在当前应用工作负载下对 SSD 缓存的压力,为 SSD 缓存负载均衡提供依据.

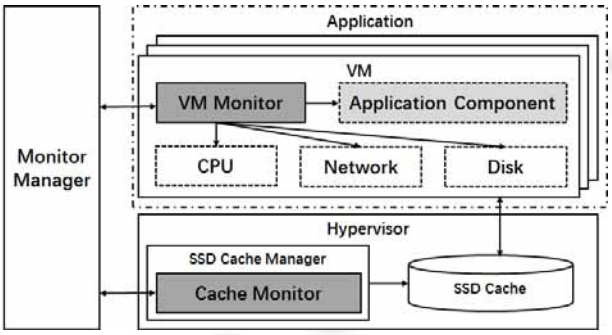

Fig.4 Fine-Grained monitoring of SSD caching and application

图 4 针对 SSD 缓存和应用的细粒度监测

在应用需求方面,我们针对 Hadoop 和 ZooKeeper 这两类主流应用设置了特定的先验规则,以帮助感知应用对虚拟机放置的倾向性.

对 Hadoop 应用而言,我们通过监测 Java 进程来确定 Hadoop 节点的角色,包括 HDFS 的元数据节点、备份元数据节点和数据节点,以及 YARN(yet another resource negotiator(Hadoop 资源管理器))计算框架的主控节点和计算节点,之后,我们监测计算节点使用的网络带宽以及与之交互的数据节点的 I/O 使用率确定计算节点对数据本地性的依赖程度.这一依赖程度反映了 Hadoop 应用对虚拟机内聚的倾向.此外,对 Hadoop 集群存在先验的虚拟机隔离需求,即 Hadoop 的元数据节点和备份元数据节点倾向于分开,而数据节点倾向于均匀分布,这可以最大限度地满足 Hadoop 对元数据可靠性的需求以及对数据副本的需求.

对 ZooKeeper 应用而言,我们通过分析每个 ZooKeeper 节点上的运行日志确定 ZooKeeper 的执行状态,包括当前 Leader 选举的状态、Quorum 状态以及每个节点的角色(leader,follower 和 observer).基于 Quorum 信息,监控模块可以确定集群能够正常运行的最小规模,这反映了应用对虚拟机隔离的倾向性,亦即放置虚拟机时倾向于分开到尽可能多的 Hypervisor 上,保证 Hypervisor 在出现单点失效时存活的 ZooKeeper 节点依然能够正常提供服务.由于 ZooKeeper 需要同步节点的读写操作、实现数据存储以及对数据状态的维护,因此也存在一定的虚拟机内聚的倾向性.与 Hadoop 应用类似,这一虚拟机内聚倾向性可以通过分析各节点的网络带宽和 I/O 使用率得出.

本文主要关注 Hadoop 和 ZooKeeper 两类应用.对于其他类型的应用,需要提供相应的先验规则以刻画应用对虚拟机放置的倾向性.但同时,系统对于没有设置先验规则的应用也设计了默认策略.这一策略基于一个简单假设,即 I/O 依赖越强的虚拟机越倾向于放置在同一 Hypervisor 上.其主要思路是:分析应用中的每台虚拟机的本地 I/O 负载以及这一虚拟机和其他虚拟机之间通过网络的数据交互带宽,两者的比值确定了虚拟机之间的依赖程度,即对虚拟机内聚的倾向性.可以直观地分开依赖程度较低的虚拟机,而将依赖程度较高的虚拟机放置在同一 Hypervisor 上.

3.3 基于聚类的虚拟机放置方案生成

我们提出了基于聚类的虚拟机放置方案的生成方法,将所有会对虚拟机放置产生影响的因素作为特征向

量的维度进行考虑,最终得出虚拟机的优化放置方案.这一方法基于 *K*-Means 算法,最终目标是将集群中的所有虚拟机聚成 *X* 类,*X* 为集群中的 Hypervisor 的数量.集群中每台虚拟机都会由一个独立的特征向量进行刻画.

在生成特征向量时,我们需要对监测模块收集到的数据进行预处理操作.如前文所述,我们监测到了虚拟机集群中的应用边界以及应用中各虚拟机的关联.这天然决定了某些虚拟机的特征向量属于同一组,这些虚拟机之间具有一定的相似性,且对放置有类似的倾向.此外,对监测到的 I/O 性能数据和 SSD 缓存使用状况的数据需要进行归一化操作,确保最终聚类结果在满足 SSD 服务能力约束的情况下,最大限度地使用 SSD 资源.

我们使用如下维度来构成特征向量,这些特征值都归一化到 0~1 的数据范围内.这里,我们以 Hadoop 为例解释这些特征的具体表现.

(1) 虚拟机 I/O 性能:主要包括虚拟机的 CPU 处理时间中的 I/O 时间、磁盘读写带宽和读写 IOPS.这些指标反映了虚拟机在应对当前工作负载时的 I/O 压力.例如,对于 Hadoop 中的 HDFS 数据节点而言,其平均 I/O 时间相对较高,且集中在顺序读写操作;对于 HDFS 元数据节点而言,I/O 负载主要集中在随机读写操作.这一特性可以区分不同 I/O 访问频率的虚拟机.

(2) SSD 缓存特性:主要包括缓存容量、缓存利用率、读写操作数和读写命中率.这些指标反映了虚拟机对缓存的使用情况.其中,缓存利用率和缓存命中率直观反映了虚拟机对缓存的依赖程度和缓存的性价比.SSD 对于随机读写的提升比顺序读写更为明显.以 Hadoop 为例,HDFS 的数据节点的缓存读写命中率会比 HDFS 元数据节点要低,同时占用更多的缓存空间.这一特性可以区分不同访问模式的虚拟机.此外,缓存容量是由 SSD 缓存系统中的容量规划组件根据虚拟机的 I/O 特征和应用的行为确定的,不在本文的讨论范围内.

(3) 网络特征:主要包括虚拟机对网络带宽的使用情况,包括与从属应用相关的连接数以及这些连接占用的网络带宽.这些指标反映了虚拟机对应用内其他虚拟机的依赖程度,间接反映了虚拟机的内聚倾向.对 Hadoop 应用而言,计算节点(YARN)到存储节点(HDFS)的网络带宽较高,建立的连接数也相对较多,这说明计算节点对存储节点存在一定的依赖.这一特性可以区分不同内聚倾向的虚拟机.

(4) 虚拟机的内聚程度:主要由虚拟机的网络特征和所依赖目标虚拟机的 I/O 性能综合计算得出.在计算虚拟机内聚程度时,我们将 0~1 的数据范围等分为 *X* 份(*X* 为 Hypervisor 数量),并将每个应用中所有虚拟机按照对应的内聚关系分组,最后将这些分组中的虚拟机的内聚程度放置在特定的 *X* 等分数据范围内,并在数据中心点周围随机取值.内聚程度越高的虚拟机组,这一值的方差越小,即偏移的随机量越小.对 Hadoop 应用而言,对特定存储节点依赖程度较高的计算节点到这一存储节点的偏差相对较小,而与不相关的存储节点偏差较大,从而体现虚拟机的内聚倾向.

(5) 虚拟机的隔离程度:这一指标主要基于先验知识.与处理虚拟机的内聚程度类似,我们将 0~1 的数据范围等分为 *X* 份(*X* 为 Hypervisor 数量),之后以应用为单位,将需要隔离的虚拟机组(如 Hadoop 集群中的元数据服务器和备份元数据服务器、ZooKeeper 中的 Quorum)分到不同的数据范围内,并在对应的数据中心点周围随机取值.对于其他类型的应用,使用第 3.2 节所述的默认策略确定隔离程度.

在执行具体的聚类算法时,我们做了如下改进.

(1) 在选取初始质心时,我们并非随机选取,而是选取了所有应用中虚拟机隔离程度最高的最多 *X* 台虚拟机(*X* 为 Hypervisor 数量),这一初始质心选取方案将会减少质心的调整次数,从而帮助算法快速收敛到最终结果.

(2) 在进行迭代的聚类操作时,我们额外考察了每个聚类中的虚拟机的 I/O 负载以及对缓存容量、SSD 带宽以及 IOPS 的需求,以满足 SSD 缓存供给能力的约束.这一改进确保最终生成的虚拟机放置方案不会导致虚拟机对 SSD 缓存资源的争用.

(3) 聚类算法的具体参数中包含对每个聚类的元素上下限的设置,控制每个聚类中包含的元素个数,从而防止降低数据倾斜的概率.

最终,聚类算法会生成优化的虚拟机放置方案.需要注意的是,在进行特征选取和聚类算法计算时,我们并

没有考虑虚拟机之间的网络延迟,这是由于在虚拟化环境下,物理服务器通常位于同一个数据中心,物理服务器间以及虚拟机间的网络延迟相比迁移的开销而言可以忽略.

3.4 迁移方案生成

虚拟机的动态迁移是代价相对较高的操作,特别是在 SSD 缓存系统中,需要在进行动态迁移时移动缓存中的脏数据,对性能的影响更为明显.因此,在通过聚类算法得到优化的虚拟机放置方案之后,我们需要计算出最少的迁移步骤以及优化的迁移顺序,尽可能地降低迁移操作对运行时虚拟机的影响.

- 首先,聚类算法的结果并不包含聚类到主机的映射,我们需要根据当前 Hypervisor 上的虚拟机放置情况找出 Hypervisor 到聚类的最大匹配,从而移动最少数量的虚拟机即可转变到最优放置方案.
- 其次,在得到 Hypervisor 到聚类的映射的基础上,我们需要匹配虚拟机的迁移操作并最终给出迁移操作的序列.
- 最后,我们需要根据先验知识确定开销最小的虚拟机迁移顺序.

Hypervisor 集群上当前所有虚拟机的放置状态 P_0 可以表达为一个集合:

$$P_0=\{H_1,H_2,\ldots,H_n\},$$

其中,

$$H_i=\{VM_{i_1},VM_{i_2},...,VM_{i_k}\}$$

P_0 集合中,元素 H_i 代表不同的 Hypervisor, VM_{i_k} 代表在 H_i 上放置的虚拟机.

聚类结果 C 也可以表达为一个集合:

$$C=\{c_1,c_2,\ldots,c_n\},$$

其中,

$$c_i=\{VM_{i_1},VM_{i_2},...,VM_{i_k}\}.$$

C 集合中的元素 c_i 代表不同的聚类, VM_{i_k} 代表属于聚类 c_i 的虚拟机.

我们的目标是找到一个 H_i 到 c_i 的映射,使得移动最少的虚拟机即可将放置状态 P 变为理想放置状态.

查找最优映射的方法 CalculateOptimizedMapping 如图 5 所示,其主要原理是查找所有映射可能的映射方案并计算其中虚拟机的匹配度,其算法复杂度为 $O(nm)$,其中,n 为聚类集合数量,m 为集合中元素个数.其中使用到的函数 FindMapping 查找所有可能映射方案,即找到主机到聚类的全排列方案.考虑到 Hypervisor 个数相对较少,本文中 FindMapping 函数直接使用了递归求解,其算法复杂度为 $O(n!)$.函数 CalcualteFitness 用于计算特定映射方案的匹配度,其算法复杂度为 $O(n^2)$.最终,CalculateOptimizedMapping 遍历所有可能的匹配并返回最优匹配方案.

```
算法 1. CalculateOptimizedMapping.
输入:当前虚拟机放置方案P0;聚类结果C.
输出:物理机到聚类的映射 M.
mappings←FindMapping(P0,C);
currentMaxFitness←0;
foreach mapping m in mappings do
    fitness←CalculateFitness(m);
    if fitness>currentMaxFitness then
        currentMaxFitness←fitness;
        M←m;
    end
end
return M;
```

Fig.5 Calculate optimized mapping of hypervisor and cluster

图 5 计算的主机到聚类的最优映射

计算特定映射的虚拟机匹配度的算法如图 6 所示.计算匹配度的过程是对放置方案中的每一个 Hypervisor 上的虚拟机集合,计算其对应的聚类中的虚拟机集合的交集的秩.映射的虚拟机匹配度即为所有对应交集的秩的总和.其算法复杂度为 $O(n^2)$.

在计算完 Hypervisor 到聚类的最优映射之后,我们需要计算从当前虚拟机放置方案到目标聚类的虚拟机

迁移操作集合.

在这里,我们定义两个原子操作.

(1) 迁入虚拟机:$H_i.In(VM_j)$,代表将虚拟机 VM_j 从其他 Hypervisor 迁入 H_i.

(2) 迁出虚拟机:$H_i.Out(VM_j)$,代表需要从 Hypervisor H_i 中迁出虚拟机 VM_j.

```
算法 2. CalculateFitness.
输入:物理机到聚类的映射 M.
输出:虚拟机匹配度 f.
f←0;
foreach VMSet s0 in M.host do
    VMSet s1←M.cluster
    f←f+count(intersect(s0,s1));
end
return f;
```

Fig.6 Calculate fitness of the mapping

图 6 计算映射的匹配度

计算迁移操作的算法如图 7 所示,其主要原理是:比较原始放置方案和目标聚类结果中对应的虚拟机集合,需要迁出的虚拟机即为原始虚拟机集合与交集的差集;需要迁入的虚拟机即为聚类结果中的虚拟机集合与交集的差集.

```
算法3. CalculateMigrationOperations.
输入:物理机到聚类的映射 M.
输出:迁移原子操作集合 OpSet.
OpSet←Empty;
foreach VMSet s0 in M.host do
    VMSet s1←M.cluster
    Set outSet←s0−intersect(s0,s1);
    Set inSet←s1−intersect(s0,s1);
    foreach VM vm in outSet do
        OpSet.add(M.host.Out(vm));
    end
    foreach VM vm in inSet do
        OpSet.add(M.host.In(vm));
    end
end
return OpSet;
```

Fig.7 Calculate migration operations

图 7 计算迁移操作

之后,我们匹配对应的迁移原子操作以生成迁移方案.算法如图 8 所示,这一算法匹配了迁移原子操作集合中从属于同一个虚拟机的迁入和迁出操作.

最后,我们将根据先验规则,配合监测到的虚拟机的 I/O 负载以及虚拟机在应用中的交互,最终确定迁移顺序.基本原则如下.

(1) 针对虚拟机持续监测,优先迁移 I/O 负载较低的虚拟机.较低的 I/O 负载意味着虚拟机目前处于相对空闲的状态,执行动态迁移所带来的性能影响会相对较小.

(2) 针对虚拟机持续监测,优先迁移对缓存依赖程度较低的虚拟机.这些虚拟机的缓存脏数据较少,可以以相对较快的速度完成动态迁移操作.

(3) 针对应用持续监测,优先迁移被依赖程度较低的虚拟机.被依赖程度较低意味着对虚拟机有数据本地性需求的其他虚拟机的数量较小,迁移这一虚拟机并不会对其他虚拟机的性能产生较为严重的影响.

(4) 持续监测网络带宽用量,限制虚拟机动态迁移的带宽.这一规则限制了虚拟机动态迁移的速度,以牺牲迁移效率为代价降低对性能的影响.考虑到应用的工作负载是长期的、具有周期性的,低开销水平

的动态迁移是可以接受的.

最终生成的优化迁移顺序会交由具体的虚拟机动态迁移模块完成迁移操作.在迁移虚拟机时,会持续监测虚拟机的性能和当前负载,在虚拟机负载较低时执行迁移,从而降低迁移操作对虚拟机性能的影响.

```
算法 4. MatchMigrationOperations.
输入:迁移原子操作集合 OpSet.
输出:迁移方案 Plan.
Plan←Empty;
foreach AtomOperation op in OpSet do
    if op.visited==false && op.type==Out then
        foreach AtomOperation anotherOp in OpSet do
            if (op!=anotherOp)
                && (anotherOp.type==In)
                && (op.VM==anotherOp.VM) then
                matchedOp←anotherOp;
                break;
            end
        end
        Plan.add(op.VM to matchedOp.VM);
        op.visited←true;
        matchedOp.visited←true;
    end
end
return Plan;
```

Fig.8 Match migration operations

图 8 匹配迁移操作

3.5 系统实现

我们在 Xen Hypervisor[16]上实现了虚拟化环境下面向多目标优化的自适应 SSD 缓存原型系统.系统架构如图 9 所示.

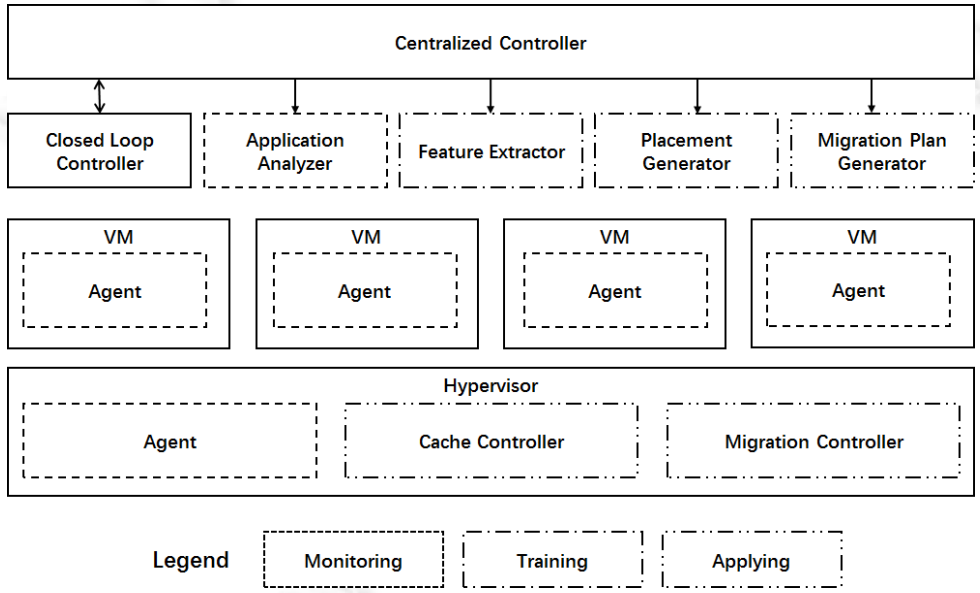

Fig.9 System architecture

图 9 系统架构

按照自适应闭环中的角色来划分,系统中的主要功能模块如下.

- 闭环控制部分

(1) 中央控制器:中央控制器用于协调整个自适应闭环的执行.中央控制器开放了 RESTful 风格的 API 供管理员跟踪系统的执行状况,并支持与虚拟化管理程序的深度整合.

(2) 闭环控制器:闭环控制器持续调用监测模块,基于滑动窗口机制,比较上一个窗口和当前窗口中持续监测的虚拟机数据的变化情况确定是否触发新一轮的调整操作.当感知到虚拟机集群规模发生变化、应用的边界或应用内虚拟机之间的关联发生变化,或是通过先验知识初步得出的应用的虚拟机放置倾向性发生变化时,会开始新一轮的闭环执行.

- 监测部分

(1) 部署在虚拟机上的代理(agent)模块:部署在虚拟机上的代理模块通过串口和 Xen Hypervisor 进行通信.Xen Hypervisor 会将虚拟机中的串口映射到 Hypervisor 的特定 Socket 上,从而允许通过 TCP 协议进行交互.代理模块开放了专有协议,允许其他组件执行具体的监测和维护指令来获得虚拟机内部的性能信息,主要包括 CPU、内存、磁盘 I/O 以及网络 I/O 的使用情况.此外,部署在虚拟机上的代理模块也会执行相应的维护命令来确定虚拟机在应用中的角色、虚拟机与应用中其他虚拟机的交互,并对应用的执行日志进行分析,以感知应用对虚拟机放置的倾向性.

(2) 部署在 Hypervisor 上的代理(agent)模块:由于 SSD 部署在每个 Hypervisor 上,SSD 缓存本身是对虚拟机透明的,无法在虚拟机内部监测到,需要由部署在 Hypervisor 上的代理模块处理.这一模块主要用于监测 SSD 缓存状态,包括缓存大小、缓存利用率、读写操作数以及读写命中率等.

(3) 应用探查模块:应用探查模块主要用于控制部署在虚拟机和 Hypervisor 上的代理模块,收集应用相关的性能数据、虚拟机在应用中的角色以及虚拟机之间的交互,结合从代理模块中获得的虚拟机 I/O 性能和 SSD 缓存状态数据,最终确定虚拟机集群中的应用边界以及应用对虚拟机放置的倾向性,为聚类算法的运行提供支持.

- 训练部分

(1) 特征抽取模块:特征抽取模块主要收集监测模块传递的虚拟机 I/O 性能数据和 SSD 缓存状态数据,以及应用探查模块传递的应用状态以及对虚拟机放置的倾向性数据,并以虚拟机为单位将其加工成特征向量,供聚类算法调用.

(2) 放置方案生成模块:从特征抽取模块中获得所有虚拟机的特征向量,并执行 *K*-Means 聚类算法,生成具体的放置方案.

- 执行部分

(1) 迁移方案生成模块:调用 Xen Hypervisor 提供的 API 获得集群中所有 Hypervisor 上部署的虚拟机,得到当前的虚拟机放置状态;之后,根据前文所述算法生成具体的虚拟机迁移操作集合,并按照先验规则确定迁移的优先级.

(2) 缓存控制器:缓存控制器部署在每个 Hypervisor 上,用于执行实际的缓存分配操作,并在虚拟机动态迁移过程中提供缓存迁移支持,包括缓存脏数据管理以及缓存设备的添加和删除操作,保证虚拟机在完成动态迁移操作后缓存立即可用.

(3) 动态迁移模块:部署在每个 Hypervisor 上,维护 Hypervisor 上的虚拟机状态,并执行具体的虚拟机动态迁移操作.

4 实验分析

我们在两台 Hypervisor 上进行实验,Hypervisor 的硬件配置见表 1.我们部署了一个由 6 台虚拟机组成的集群,每台虚拟机使用 2 个虚拟 CPU(vCPU)和 2GB 内存,并配置了基于 SSD 缓存的存储.考虑到本节中的实验场景,我们为每台虚拟机分配了 16GB 的 HDD,配以 256MB 的 SSD 缓存.

我们选取了 I/O 基准测试、Hadoop 应用和 ZooKeeper 应用这 3 个场景来验证我们的原型系统的有效性.

在这 3 个实验场景中,CPU、内存、网络带宽和 HDD 空间都不存在瓶颈,唯一的瓶颈是 SSD 缓存.在进行实验时,将我们的方法与随机迁移方法进行了对比.随机迁移方法作为基准的对照算法,并不会感知虚拟机的状态,其主要操作流程是每经过固定的时间间隔就随机选择集群中的一台虚拟机,并将其动态迁移至另一个 Hypervisor 上,并同时迁移 SSD 缓存.对于 I/O 基准测试、Hadoop 应用以及 ZooKeeper 应用这 3 个场景均进行了 5 组测试并选取了平均值,以尽可能地消除随机算法带来的偶然性,同时保证对照组的可信度.

Table 1 Hardware configuration of Hypervisors

表 **1** Hypervisor 硬件配置

项目	描述
CPU	Intel Xeon CPU E5-2620
内存	32GB
SSD	Intel SSD 535 240GB
共享存储	通过 iSCSI 连接到后端分布式存储

考虑到虚拟机动态迁移方法的基本目标是保证所关注的资源尽量不产生争用,在我们的实验场景中,现有的以 CPU、内存、网络带宽和 HDD 空间作为目标的虚拟机动态迁移方法与随机迁移方法会有类似的表现;相比之下,去除了异常数据点的随机迁移方法涵盖了更多情形,更适合作为对照方法.

在实验过程中,我们动态地切换了虚拟机集群上的应用类型,并进一步切换了 Hadoop 的工作负载,自动触发了自适应闭环的执行,计算新的优化迁移方案,并自动触发了虚拟机的动态迁移操作.下文所述的 I/O 基准测试以及针对大数据处理应用和分布式协同应用的测试的实验结果是迁移过程稳定之后的时间片段内的结果.为公平起见,我们将随机迁移的时间间隔以及闭环控制器的时间窗口设置为 1 分钟.

4.1 I/O基准测试

首先,我们选取了 I/O 基准测试 fio,用来验证我们的 SSD 缓存系统是否能够自适应地感知 I/O 负载并实现 SSD 缓存资源的负载均衡.为模拟极端的随机读写测试场景,我们使用 fio 以 32KB 块大小在每台虚拟机上并发测试随机读写 1GB 大小文件,并限制虚拟机对内存缓冲区的使用,以最大限度地使用 SSD 缓存的供给能力.如前文所述,我们为每个虚拟机分配了 256MB 缓存,远小于测试场景中 1GB 的总工作集大小,这也可以充分触发缓存置换和数据同步操作,模拟实际的 I/O 密集型负载场景.

随机读测试的实验结果如图 10 所示,随机写测试的实验结果如图 11 所示.图中横轴表示虚拟机节点的编号,柱状图的纵轴表征了各节点上的 IOPS.折线图的纵轴表征了与随机迁移虚拟机相比,我们的方法在 IOPS 性能上的比值.

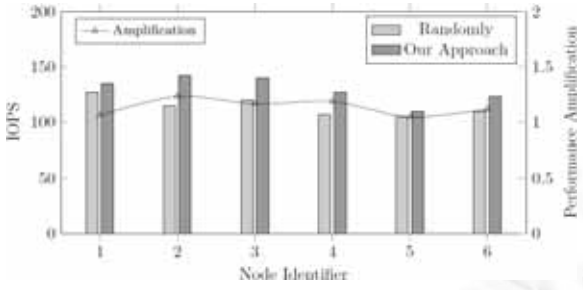

Fig.10 Random read performance

图 10 随机读测试的性能

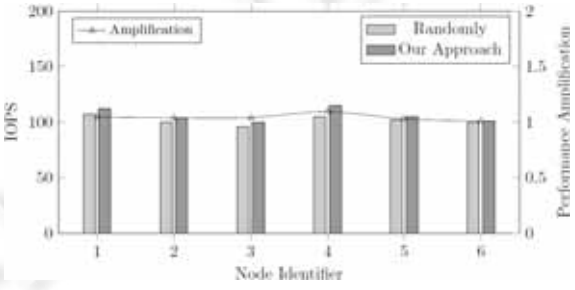

Fig.11 Random write performance

图 11 随机写测试的性能

可以看出,与随机迁移虚拟机相比,我们的迁移方案考虑到虚拟机的 I/O 负载,将高负载的虚拟机迁移到了不同的 Hypervisor,从而在随机读测试的 IOPS 方面达到了平均 15%的性能提升;在随机写测试方面,受限于缓存容量以及缓存策略,会频繁触发与后端分布式存储系统的同步操作,因此在这一场景下,只能带来平均 6%的性能提升.

4.2 大数据处理应用的性能

我们首先选取了 Hadoop 自带的一系列微基准测试(Micro benchmarks)来评价我们的迁移方法在应对 Hadoop 任务时的性能表现.我们选取了一组 CPU 敏感型的应用 PI-10 和 PI-20,运行 Hadoop 自带的 pi 基准测试,即使用 Quasi-Monte Carlo 方法计算圆周率.PI-10 代表采用了 10 个 Map 任务,PI-20 代表采用了 20 个 Map 任务.我们同时选取了一组 I/O 敏感型应用:顺序写(write)、顺序读(read)、随机读(randr)、反向读(backr),分别取自 Hadoop 自带的 TestDFSIO 基准测试.

评价 Hadoop 任务执行性能的主要指标是执行时间,实验结果如图 12 所示.图中横轴代表 Hadoop 的微基准测试的类型,柱状图纵轴表征了测试的执行时间,折线图纵轴表示了我们的方法相对于随机迁移而言在执行时间方面的减少比例.PI-10 和 PI-20 的实验结果表明:我们的方法在应对 CPU 敏感型应用场景时,性能与随机迁移类似;对于 I/O 敏感型应用而言,我们的方法考虑到的是 Hadoop 的数据本地性,因此在执行时间方面比随机迁移方法平均降低了 25%.

此外,我们还评价了 I/O 敏感型应用的吞吐率,实验结果如图 13 所示.柱状图的纵轴表征了执行对应测试的吞吐率,折线图纵轴表征了我们的方法与随机迁移方法相比吞吐率的比值.值得注意的是,这一吞吐率实际表征的是在测试执行过程中总共需要操作的数据量与实际操作数据时间的比值,实际操作数据的时间会远小于执行时间.从吞吐率角度看,我们的方法比随机迁移平均提升了 39%,对读操作和随机读操作提升更为明显.

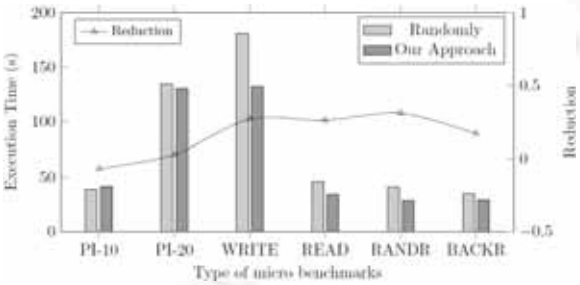

Fig.12 Execution time of Hadoop benchmarks

图 12 Hadoop 基准测试的执行时间

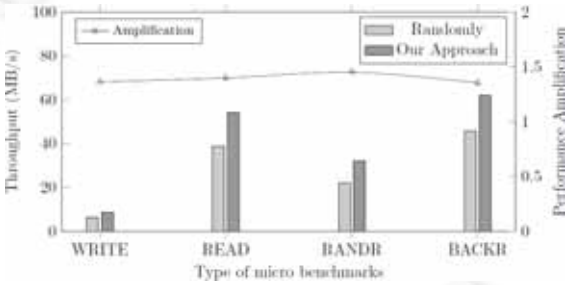

Fig.13 Throughput of Hadoop I/O benchmarks

图 13 Hadoop I/O 基准测试的吞吐率

4.3 分布式协同应用的可靠性

我们采用了逻辑距离这一简单的指标来表征分布式协同应用 ZooKeeper 的可靠性.我们定义:当两台虚拟机位于同一 Hypervisor 上时逻辑距离为 0,当位于不同 Hypervisor 上时逻辑距离为 1.集群的逻辑距离为集群中所有节点两两之间逻辑距离之和.当某一 Hypervisor 出现单点失效时,会导致位于这一 Hypervisor 上所有的虚拟机(即逻辑距离为 0 的虚拟机)宕机,而对其他虚拟机没有影响.因此,集群的逻辑距离越大,可以间接推断出集群本身的可靠性越高.

我们比较了随机迁移方法和我们的方法,在达到相对稳定状态时,其逻辑距离分别为 5 和 9.我们的虚拟机迁移方法最终将虚拟机平均分布在 2 台 Hypervisor 上,从而保证在 Hypervisor 出现单点失效时,受影响的虚拟机数量最小.

我们还比较了 ZooKeeper 应用的性能.我们选取了 ZooKeeper 的作者 Hunt 开发的 ZooKeeper Smoketest[17],主要用于评价 ZooKeeper 主要操作的延迟.我们测试了访问 ZooKeeper 集群中所有 6 个虚拟机节点时的异步操作速度,实验结果如图 14 所示.图中横轴表征了操作类型,主要包括创建永久(permanent)节点(CREATE)、写入数据(SET)、读取数据(GET)、删除永久节点(DELETE)、观察节点(WATCH).图中柱状图纵轴表征了操作速度,数值为每秒可执行的操作数,折线图纵轴表征了我们的虚拟机迁移方法与随机迁移相比在操作速度上的比例.实验结果表明:我们的虚拟机迁移方案相比随机迁移方案具有类似的性能,平均性能差距为 5%.这一性能差距,在保证 ZooKeeper 可靠性的前提下是可以接受的.

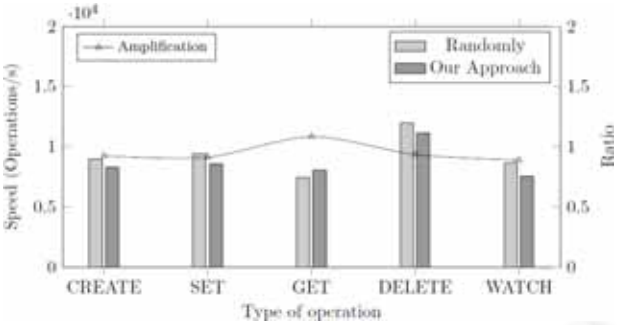

Fig.14 Performance of ZooKeeper—Async operations

图 14 ZooKeeper 异步操作性能

4.4 迁移开销

在实验过程中,我们持续测量了虚拟机动态迁移的时间以及动态迁移执行过程中的性能干扰.我们测量到单台虚拟机的动态迁移时间在 10s~15s 之间,对网络带宽的占用平均为 40%.我们测量了在虚拟机动态迁移过程中执行的 Hadoop 计算密集型和 I/O 密集型应用的任务执行时间.受益于优化的迁移顺序以及对迁移速度的控制,对 Hadoop 任务的执行时间的影响平均为 7%.这一额外开销在大规模长时间的工作负载的场景下是可以接受的.

5 相关工作

现有的缓存资源管理的相关工作主要从缓存本身出发,以缓存的特点及相关关键指标驱动缓存资源管理,最终实现性能提升或降低后端存储系统负载的目的.如在 SSD 和 HDD 构成的混合存储系统中考虑 SSD 应对不同类型 I/O 负载的差异性,进行选择性的数据缓存[18]以提升性能;或使用本地磁盘介质作为网络存储系统的缓存[19],以降低网络存储系统的负载,提升系统性能.

许多研究工作考虑 SSD 缓存本身的特点进行缓存分配.RAF[20]引入成本收益模型,以缓存对 SSD 而言价值较高的随机读写数据,降低 SSD 的响应时间,并提升寿命.Centaur[6]将 Hypervisor 上的 SSD 缓存进行分区并分配给不同的虚拟机,通过各虚拟机的缓存大小,以满足诸如缓存 Miss 率或虚拟机 I/O 响应时间等具体性能指标的优化需求.S-CAVE[9]基于缓存命中率和虚拟机频繁使用的缓存空间大小导出了虚拟机对缓存空间的需求,并以此实现 SSD 缓存资源分配.vCacheShare[21]从虚拟机的 I/O 行为和特点出发,结合虚拟机的数据本地性以及缓存命中率和重用程度等特性,指导 SSD 缓存资源分配.这些工作都是以 SSD 缓存的关键指标作为调整依据,面向单台 Hypervisor 的场景,并没有从全局角度考虑 SSD 缓存资源的供给能力,也难以在 SSD 缓存出现资源争用时进行有效应对.

一些研究工作考虑了缓存资源出现争用的情况并加以解决.CloudCache[8]提出了一种基于 Reused Working Set 的 SSD 缓存方法,可以在缓存命中率相当的前提下有效降低需要缓存的数据大小,从而提高缓存的利用率.此外,CloudCache 也提出了一种以缓存容量为导向的虚拟机动态迁移策略,在虚拟机对缓存容量的需求超过 SSD 总容量时触发虚拟机动态迁移.然而,CloudCache 没有考虑到 SSD 作为存储介质本身对带宽和 IOPS 能力的约束,也没有考虑到应用对虚拟机放置的倾向性,因此,可能在满足缓存容量约束的前提下出现虚拟机对 SSD 的带宽和 IOPS 的争用,也可能在执行迁移的过程中对虚拟机进行了错误的放置,影响了虚拟机的性能或虚拟机上承载应用的可靠性.

此外,一些研究工作考虑到了应用内的工作负载,并以此指导了 SSD 缓存分配.Capo[22]面向虚拟桌面的场景,考虑到了用户虚拟机中的不同目录和文件的使用模式和对缓存的不同需求.Capo 为用户磁盘内的不同目录设置了不同的缓存策略,以此提高缓存利用率,避免不必要的缓存操作,降低存储系统的负载.文献[23]构建了一个原型文件系统,在底层对 I/O 请求进行分类,针对不同类型的 I/O 请求设置不同的缓存策略,从而实现缓存命中

率和虚拟机性能的提升.这些研究工作对虚拟机内运行的工作负载进行了建模,从而可以有效地进行应对,但它们未能从全局的角度考虑多台虚拟机组成的应用以及应用内部虚拟机关联对缓存资源管理的影响.

6 结论与未来的工作

本文提出了虚拟化环境下面向多目标优化的自适应 SSD 缓存系统,构建了一个自适应闭环,通过持续监测并感知模式变化以驱动虚拟机的动态迁移,最终在容量规划之外实现了对 SSD 缓存利用率、应用性能和应用可靠性的多目标优化.系统在监测时重点关注了虚拟机和 SSD 缓存的性能指标,并感知虚拟机上部署的应用的边界、应用内虚拟机的关联以及应用本身对虚拟机放置的倾向性;之后,使用聚类算法获得优化放置方案,并给出了合理的迁移顺序以降低开销.

这一工作的主要不足之处在于,对应用类型的覆盖范围较小,目前只完整支持 Hadoop 和 ZooKeeper 两类应用,且依赖于一定的先验知识来获得应用对虚拟机放置的倾向.此外,系统的监测粒度也相对有限,未能深入大数据处理应用内部去感知用户代码的行为,以此指导 SSD 缓存负载均衡和对应用性能和可靠性的优化.

未来我们将刻画应用行为以及使用 SSD 缓存时的共性,从这两个角度对应用进行自动分类,更合理地感知应用对 SSD 的需求,从而支持虚拟化环境下更多类型的应用.此外,还将实现更深层次的应用感知,从用户代码层面考虑应用的行为和对 SSD 缓存的使用情况,从而支持更灵活的 SSD 缓存资源管理.

References:

[1] Gulati A, Shanmuganathan G, Zhang X, Varman P. Demand based hierarchical QoS using storage resource pools. In: Proc. of the 2012 USENIX Conf. on Annual Technical Conf. Boston: USENIX Association, 2012. 1−13.

[2] Hansen JG, Jul E. Lithium: Virtual machine storage for the cloud. In: Proc. of the 1st ACM Symp. on Cloud Computing. Indianapolis: ACM Press, 2010. 15−26. [doi: 10.1145/1807128.1807134]

[3] Ye L, Lu G, Kumar S, Gniady C, Hartman JH. Energy-Efficient storage in virtual machine environments. In: Proc. of the 6th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. Pittsburgh: ACM Press, 2010. 75−84.

[4] Lu L, Pillai TS, Arpaci-Dusseau AC, Arpaci-Dusseau RH. WiscKey: Separating keys from values in SSD-conscious storage. In: Proc. of the 14th USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2016. 133−148. [doi: 10.1145/3033273]

[5] Kim J, Lee D, Noh S H. Towards SLO complying SSDs through OPS isolation. In: Proc. of the 13th USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2015. 183−189.

[6] Koller R, Mashtizadeh AJ, Rangaswami R. Centaur: Host-Side SSD caching for storage performance control. In: Proc. of the 2015 IEEE Int'l Conf. on Autonomic Computing. Wurzburg: IEEE Computer Society, 2015. 51−60. [doi: 10.1109/ICAC.2015.44]

[7] Oh Y, Lee E, Hyun C, Choi J, Lee D, Noh S H. Enabling cost-effective flash based caching with an array of commodity SSDs. In: Proc. of the 16th Annual Middleware Conf. Vancouver: ACM Press, 2015. 63−74. [doi: 10.1145/2814576.2814814]

[8] Arteaga D, Cabrera J, Xu J, Zhao M. CloudCache: On-Demand flash cache management for cloud computing. In: Proc. of the 14th USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2016. 355−369.

[9] Luo T, Ma S, Lee R, Zhang X, Liu D, Zhou L. S-CAVE: Effective SSD caching to improve virtual machine storage performance. In: Proc. of the 22nd Int'l Conf. on Parallel Architectures and Compilation Techniques. Edinburgh: IEEE Computer Society, 2013. 103−112. [doi: 10.1109/PACT.2013.6618808]

[10] Shvachko K, Hairong K, Radia S, Chansler R. The hadoop distributed file system. In: Proc. of 2010 IEEE the 26th Symp. on Mass Storage Systems and Technologies (MSST). Incline Village: IEEE Computer Society, 2010. 1−10. [doi: 10.1109/MSST.2010.5496972]

[11] Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B, Baldeschwieler E. Apache hadoop YARN: Yet another resource negotiator. In: Proc. of the 4th Annual Symp. on Cloud Computing. Santa Clara: ACM Press, 2013. 1−16. [doi: 10.1145/2523616.2523633]

[12] Junqueira FP, Reed BC, Serafini M. Zab: High-Performance broadcast for primary-backup systems. In: Proc. of the 2011 IEEE/IFIP 41st Int'l Conf. on Dependable Systems&Networks. Hong Kong: IEEE Computer Society, 2011. 245−56. [doi: 10.1109/DSN.2011.5958223]

[13] Hunt P, Konar M, Junqueira FP, Reed B. ZooKeeper: Wait-Free coordination for Internet-scale systems. In: Proc. of the 2010 USENIX Conf. on USENIX Annual Technical Conf. Boston: USENIX Association, 2010. 11–24.

[14] Intel. Intel solid state drive 750 series. 2016. http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-750-spec.pdf

[15] WD. WD black PC HD series specification sheet. 2016. https://www.wdc.com/content/dam/wdc/website/downloadable_assets/eng/spec_data_sheet/2879-771434.pdf

[16] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In: Proc. of the 19th ACM Symp. on Operating Systems Principles. Bolton Landing: ACM Press, 2003. 164–177. [doi: 10.1145/945445.945462]

[17] Hunt P. ZooKeeper smoketest. 2016. https://github.com/phunt/zk-smoketest

[18] Yang PY, Jin PQ, Yue LH. A time-sensitive and efficient hybrid storage model involving SSD and HDD. Chinese Journal of Computers, 2012,35(11):2294–2305 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2012.02294]

[19] Yin Y, Liu ZJ, Xu L. Cache system based on disk media for network storage. Ruan Jian Xue Bao/Journal of Software, 2009,20(10): 2752−65 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3427.htm [doi: 10.3724/SP.J.1001.2009.03427]

[20] Liu Y. Research on caching and prefetching technologies for hierarchical hybrid storage systems [Ph.D. Thesis]. Wuhan: Huazhong University of Science and Technology, 2013 (in Chinese with English abstract). [doi: 10.7666/d.D608789]

[21] Meng F, Zhou L, Ma X, Uttamchandani S, Liu D. vCacheShare: Automated server flash cache space management in a virtualization environment. In: Proc. of the 2014 USENIX Conf. on USENIX Annual Technical Conf. Philadelphia: USENIX Association, 2014. 133−144.

[22] Shamma M, Meyer DT, Wires J, Ivanova M, Hutchinson NC, Warfield A. Capo: Recapitulating storage for virtual desktops. In: Proc. of the 9th USENIX Conf. on File and Stroage Technologies. San Jose: USENIX Association, 2011. 3–17.

[23] Mesnier M, Chen F, Luo T, Akers JB. Differentiated storage services. In: Proc. of the 23rd ACM Symp. on Operating Systems Principles. Cascais: ACM Press, 2011. 57–70. [doi: 10.1145/2043556.2043563]

附中文参考文献:

[18] 杨濮源,金培权,岳丽华.一种时间敏感的 SSD 和 HDD 高效混合存储模型.计算机学报,2012,35(11):2294–2305. [doi: 10.3724/SP.J.1016.2012.02294]

[19] 尹洋,刘振军,许鲁.一种基于磁盘介质的网络存储系统缓存.软件学报,2009,20(10):2752−2765. http://www.jos.org.cn/1000-9825/3427.htm [doi: 10.3724/SP.J.1001.2009.03427]

[20] 刘洋.层次混合存储系统中缓存和预取技术研究[博士学位论文].武汉:华中科技大学,2013. [doi: 10.7666/d.D608789]

唐震(1990－),男,江苏苏州人,博士生,主要研究领域为网络分布式计算与软件工程.

魏峻(1970－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.

吴恒(1983－),男,博士,副研究员,CCF 专业会员,主要研究领域为网络分布式计算,软件工程.

黄涛(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.

王伟(1982－),男,博士,副研究员,CCF 专业会员,主要研究领域为网络分布式计算,软件工程.